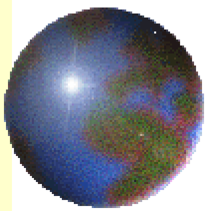
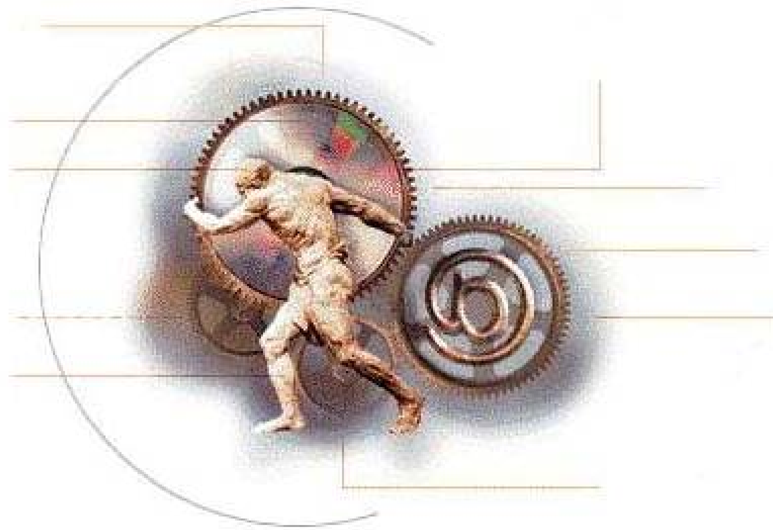


資料結構(Data Structures)

Course 6: Tree and Binary Tree



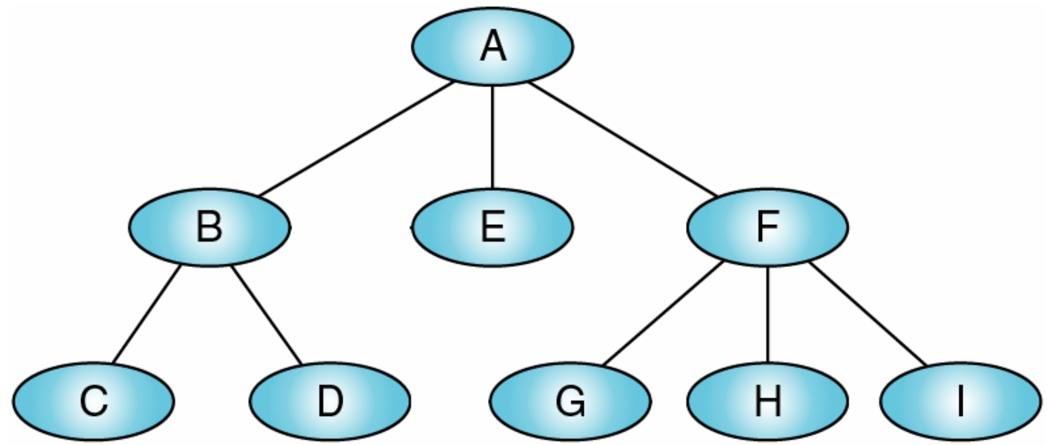
授課教師:陳士杰

國立聯合大學 資訊管理學系

Outlines

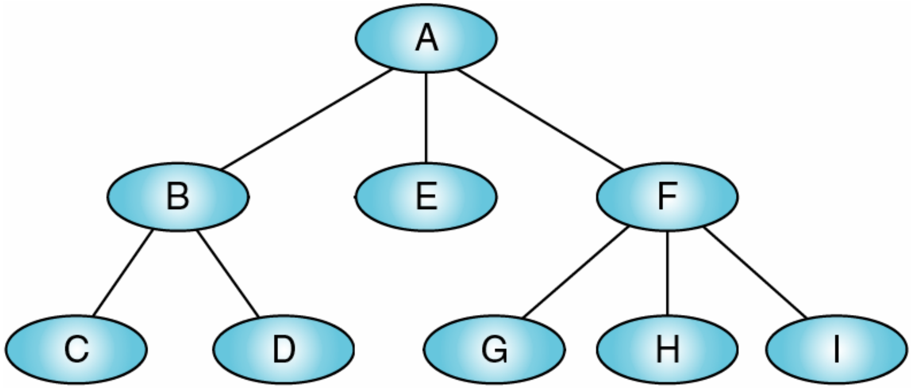
- 本章重點
  - Tree的定義、相關術語與表示方式
  - Binary Tree的定義與表示方式
  - Tree v.s. Binary Tree
  - Binary Tree三個基本定理
  - Complete Binary Tree的定理
  - Binary Tree追蹤
    - 計算
    - 演算法
    - 應用
  - Binary Tree的種類
  - Tree轉成Binary Tree
  - 引線二元樹
  - Forest轉成Binary Tree

- Def: Tree是由1個以上的Nodes所組成的有限集合，滿足:
  - 至少有一個Node，稱為Root
  - 其餘的Nodes分成T_1, T_2, …, T_n個互斥集合，稱為Subtree
- 上述定義隱含:
  - Tree 不可為空
  - 子集合(子樹)間沒有交集



Terminology (相關術語)

- **Degree of a node(節點分支度)**:
  - 節點的子樹個數
    - Degree of A = 3
    - Degree of B = 2
    - Degree of E = 0
    - Degree of F = 3
    - Degrees of C, D, G, H, I are all 0



- **Degree of a tree (樹分支度)**:
  - 樹中所有節點分支度最大者。即:Max {各Node之degree}

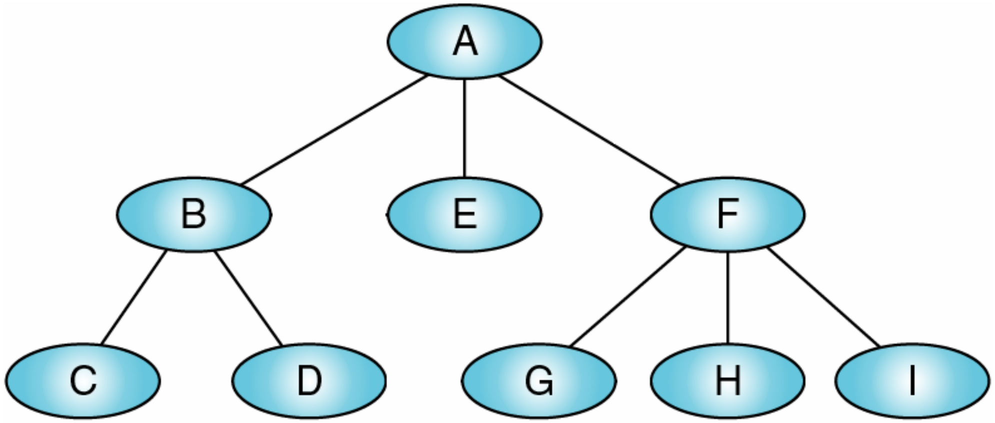
- **Leaf (葉子)**
  - 分枝度為 0 之節點。
- **Non-leaf Node (Non-terminal Node或Internal Node)**
  - 樹中所有非葉子的Node，或是Degree≥1的節點稱之。
- **Parent Node (父節點) and Child Node (子節點)**
  - 若一個節點 x 有後繼節點 (Successor Nodes)，則此節點 x 即為父節點 (Parent)；反之，若一個節點 y 有前輩節點 (Predecessor)，則此節點 y 即為子節點 (Child) 。
  - 某節點所有子樹的樹根皆為該節點的Child; 而該節點為這些樹根的Parent.

Sibling (兄弟)

- 同一個父節點的所有子節點互稱為Sibling。

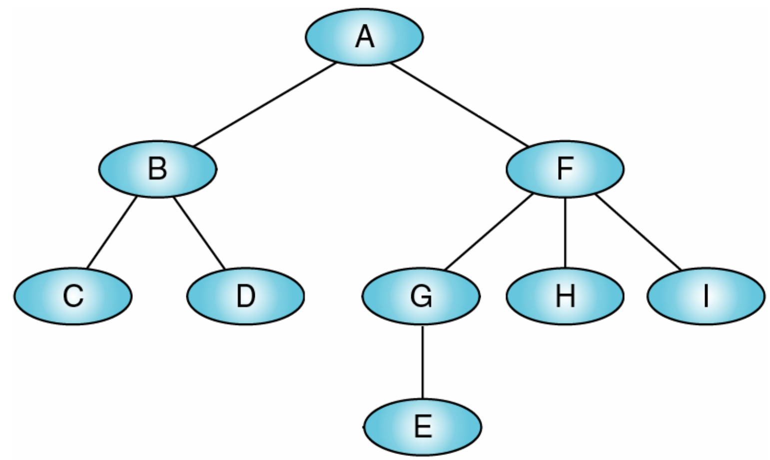
Ancestor (祖先)

- 某一個節點的祖先，乃是從樹根到該節點路徑中，所經過的所有節點。
- 通常為一**集合**
- Ancestors of C: {A, B}



Level (階度):

- 某一個節點的階度，是指自樹根至該節點的距離。
- **Root所在的level值為1**，若父點的level值為i，則子點的level值為i+1.
- 範例：
  - E的階度 = 4
  - C的階度 = 3

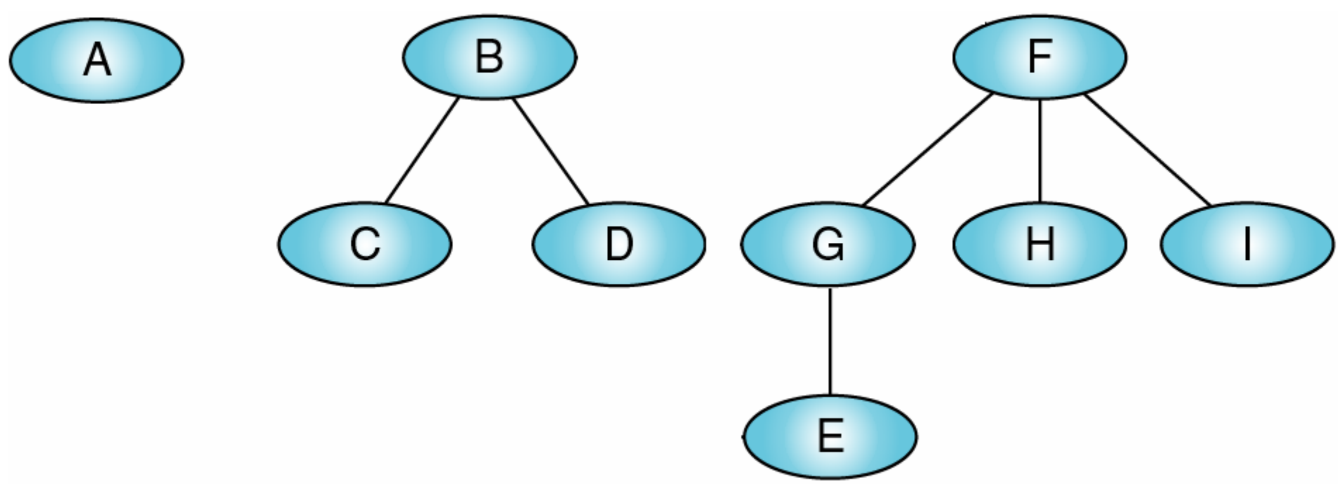


Height (高度；或稱Depth):

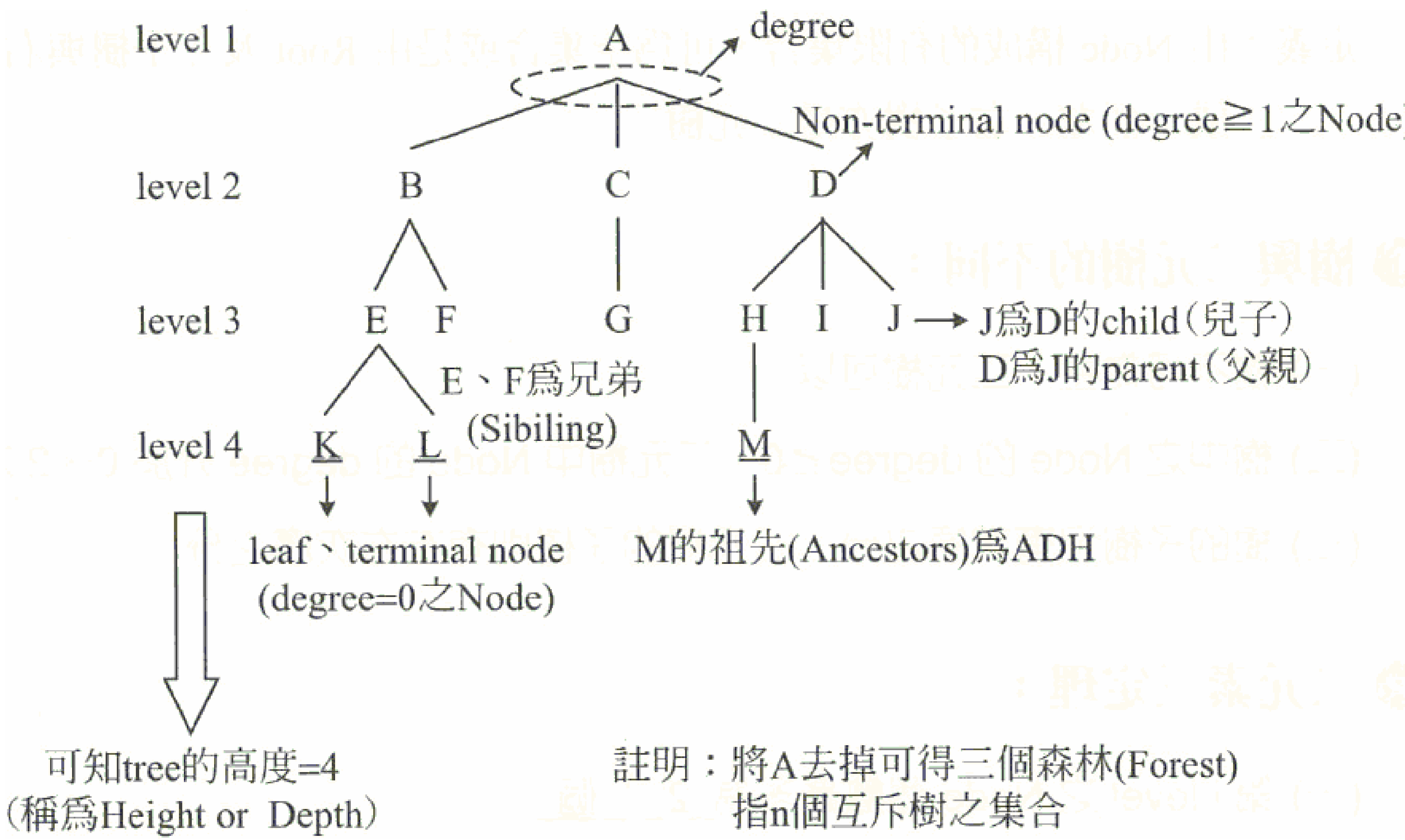
- 一顆樹的各level值當中之**最大值**，即為該樹的高度。
- 範例：
  - 此樹的高度 = 4

Forest (森林):

- 森林及是**n個互斥樹**所形成的集合 (n ≥ 0)
- 可以為空



※練習範例※



level 1
A
degree
Non-terminal node (degree≧1之Node)
level 2
B
C
D
level 3
E
F
G
H
I
J
J爲D的child(兒子)
D爲J的parent(父親)
E、F爲兄弟
(Sibiling)
level 4
K
L
M
leaf、terminal node
(degree=0之Node)
M的祖先(Ancestors)爲ADH
可知tree的高度=4
(稱爲Height or Depth)
註明：將A去掉可得三個森林(Forest)
指n個互斥樹之集合

■ Tree Representation (樹的表示方式)

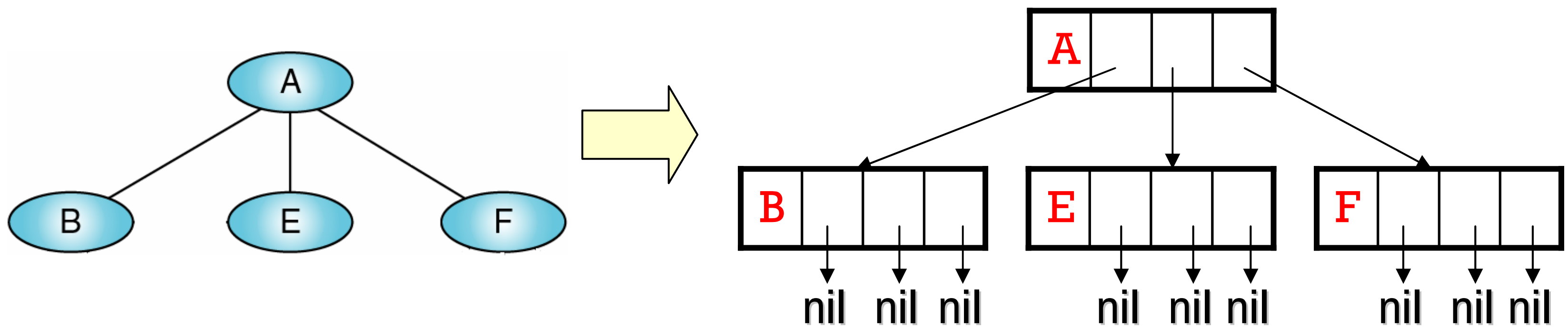
- 一顆樹要實作於電腦系統中，通常會採用**指標 (Pointer)** 來設計。
- **做法1**: **<u>直接用Link list表示</u>**
  - 假設Tree有n個node，degree為k
  - Node structure設計如下:

Data	Link 1	Link 2	…	Link k

其中:

- k: 表示Tree degree
- Data: 存node的資料值
- Link i: 指標指向 ith 子樹之Root Node ($1 \le i \le k$)

- 例如:



- **問題**: Link空間浪費甚巨 (∵空Link數目太多)

分析:

- 假設tree有**n**個nodes，tree degree為**k**
- 總共的link空間為: ____
- 有用的link數目為: ____ (即link ≠ nil)
- 浪費數目(即: 空link): ________
- 浪費比例: $\dfrac{nk-(n-1)}{nk} = \dfrac{n(k-1)+1}{nk} \cong \dfrac{k-1}{k}$

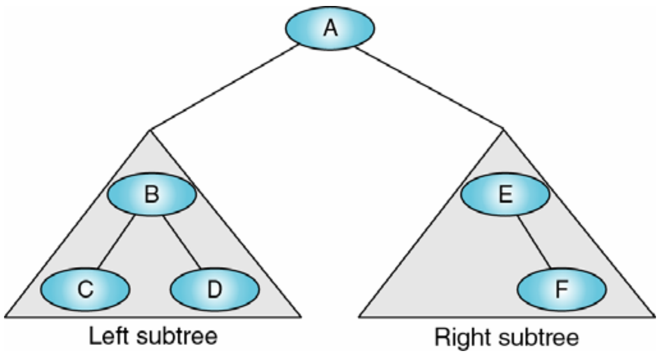
1. k愈多，**浪費比例**愈高!!
   - 若k=100，則浪費比例高達99%!!
2. 為避免上述問題，則k應**愈小愈好**。若:
   - k=1 ⇨ **鏈結串列** (∵不是tree，∴不予討論)
   - k=2 ⇨ 浪費比例**最低**

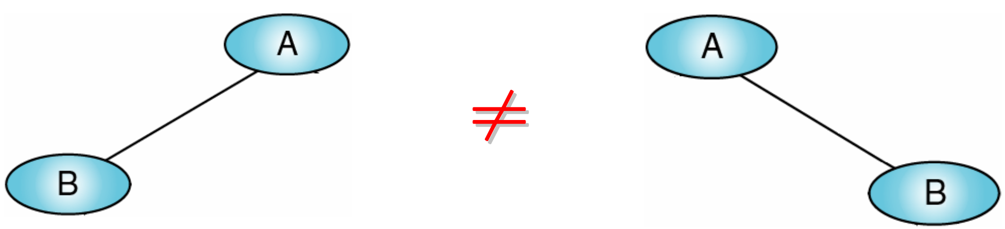
做法2: 將Tree化成Binary Tree後再存!

∴Tree化成Binary Tree是資料結構中的一個很重要的議題!!

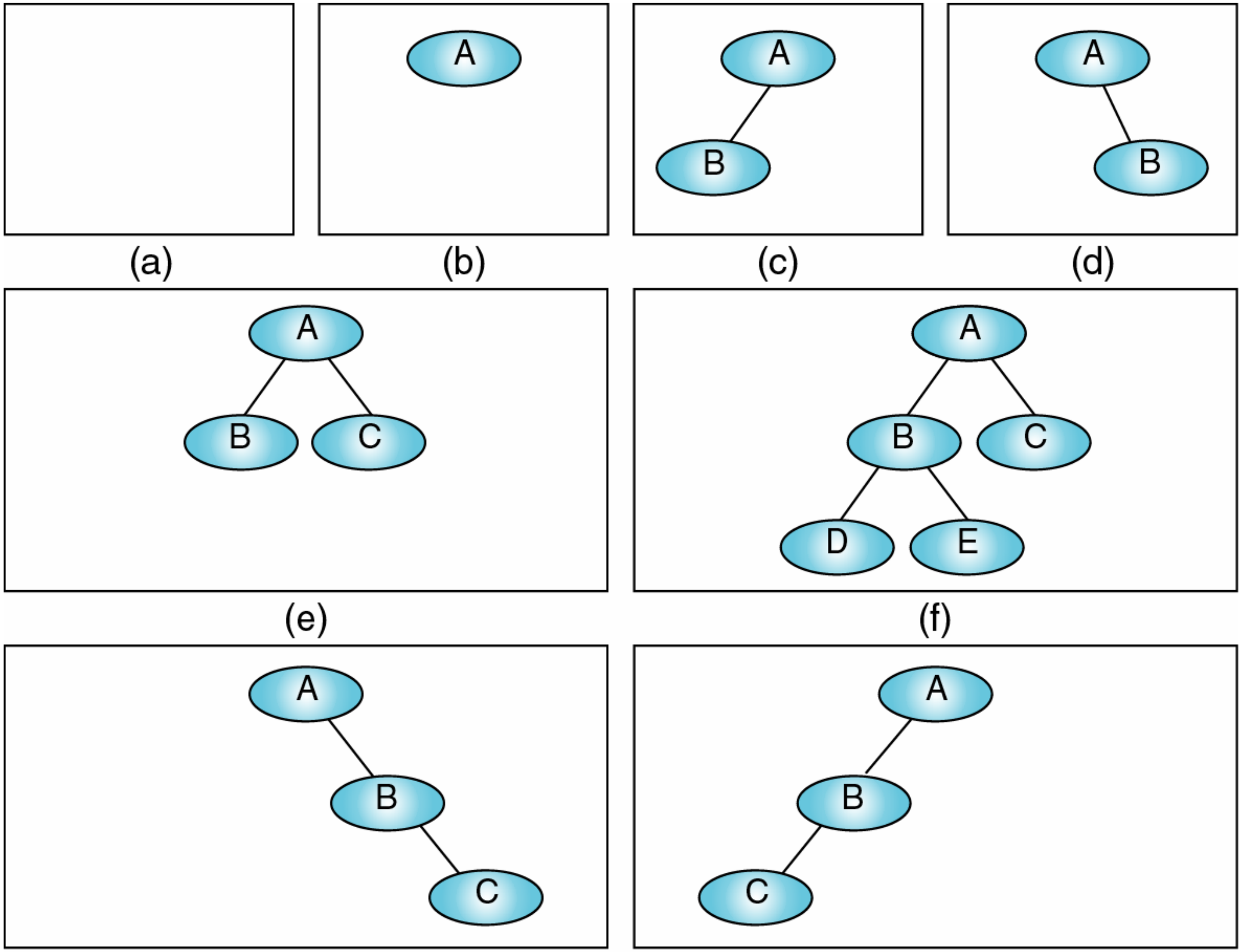
■ Binary Tree (二元樹)

- Def:
  - Binary Tree為具有 **≥ 0 個nodes**所構成的有限集合。
    - Binary Tree可以為**空的樹**。
    - 若不為空的樹，則具有Root及左, 右子樹，且左, 右子樹亦是Binary Tree。
  - 表示各node之degree介於**0與2**之間。
  - 左, 右子樹有**次序**之分
    - 故又稱**Order Tree**





- 一般樹的子樹不會去區分是左子樹、中子樹還是右子樹，∵可能的子樹型態很多!!



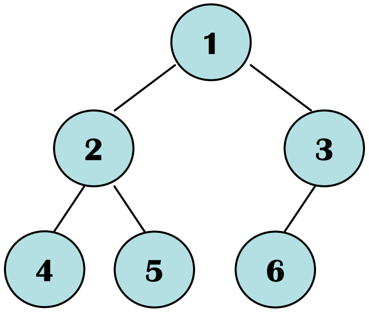
(a) (b) (c) (d) (e) (f) (g) (h)

- Tree與Binary Tree之差異:

Tree	Binary Tree

二元樹之三個基本定理

- [定理一]: 二元樹中，第i個level的node個數最多有 2^{i-1} 個。
- [定理二]: 高 (深) 度為H的二元樹，其node個數最多有 2^H-1 個 ($n_{max} = 2^H-1$)。
- [定理三]: 非空二元樹若leaf個數為 n_0 個，degree為2的node個數為n_2個，則 $n_0 = n_2+1$。



※練習範例※

- 若有15個leafs，則degree為2的node數 = ?
  - 14個
- 若有10個degree為2的nodes，則leaf個數 = ?
  - 11個
- 若二元樹有53個nodes，其中degree為1的node數有22個，則leaf個數 = ?

Sol:

$n = 53 = \mathbf{n_0+n_1+n_2}$

$= n_0+22+n_2$

$\therefore n_0 + n_2 = 53 - 22 = 31$ …①

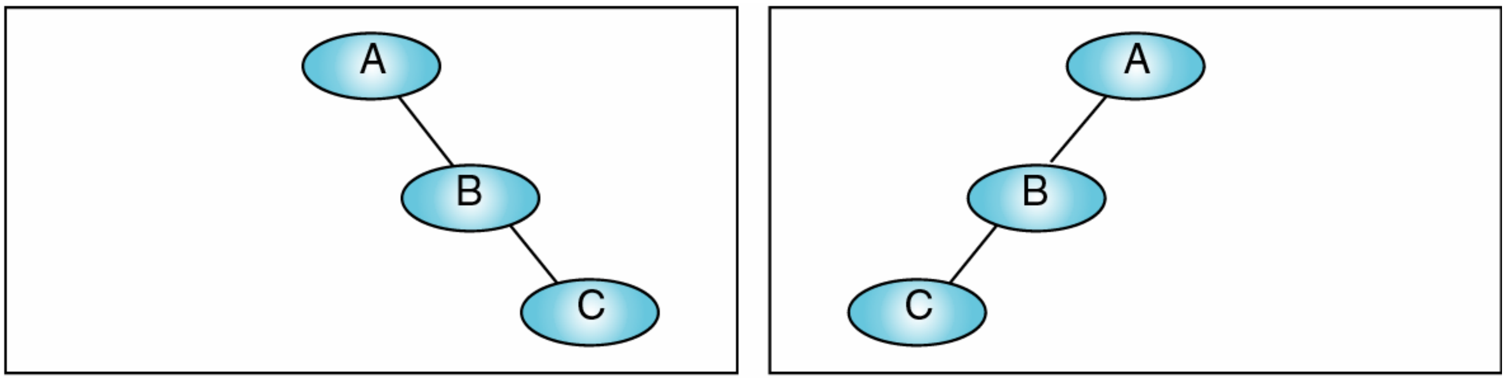
又: $n_0 = n_2 + 1$ …②

⇨ $n_0 = 16$

二元樹的種類

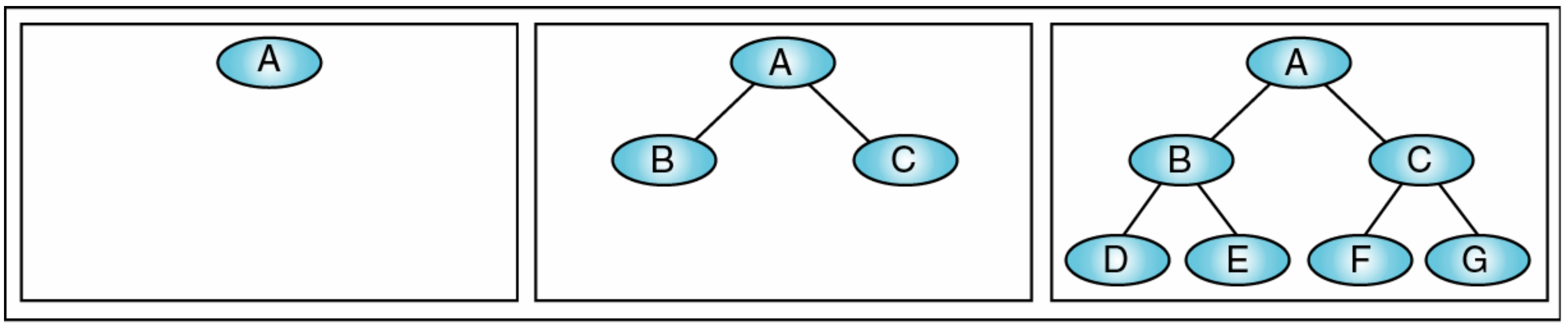
- Skewed Binary Tree (偏斜二元樹)
- Full Binary Tree (完滿二元樹)
- Complete Binary Tree (完整二元樹)

- Def: 可分為
  - Left-Skewed Binary Tree: 每個non-leaf node皆只有左子節點
  - Right-Skewed Binary Tree: 每個non-leaf node皆只有右子節點

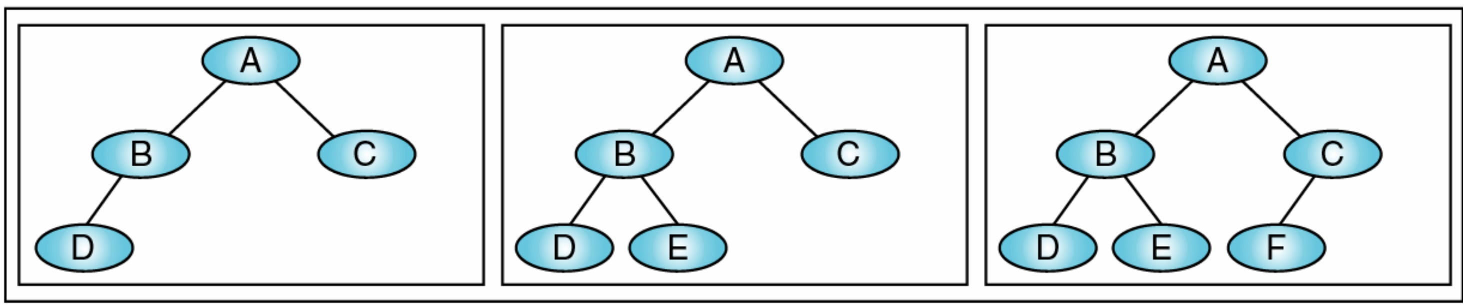


  - 此類型的tree之高度 H 為節點個數 n

- Def:
  - 具有**最多Node個數**的二元樹稱之
    - 即: 高度為 H, 其node數必為**2^H-1**
    - Full B.T. 下具有n個nodes, 其高度必為: $\lceil \log_2(n+1) \rceil$

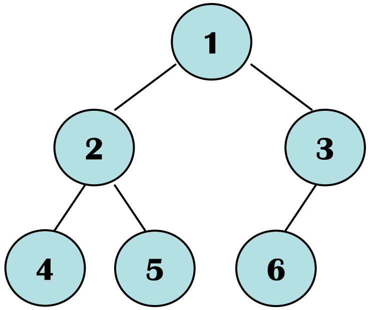


- Def:
  - 若二元樹高度為 H, node個數為n, 則
    - $2^{H-1}-1 < n < 2^{H}-1$
    - n個node之編號與高度k的full binary tree之前的n個node編號一一對應，不能跳號。



- Complete B.T. 下具有n個nodes, 其高度 H 必為: $\lceil \log_2(n+1) \rceil$

- 假設Complete B.T. 有n個節點 (編號 1~n), 其中某個節點其編號為 i, 則:
  - 其左兒子編號為 2i; 若 2i > n, 則左兒子不存在
  - 其右兒子編號為 2i+1; 若 2i+1 > n, 則右兒子不存在
  - 其父點編號為 [i/2] ([ ]: 無條件捨位取整數); 若 [i/2] < 1, 則父節點不存在
  - 範例: 有一個 6 Nodes 的 Complete B.T.:
    - Node 5 的 Parent為何？
    - Node 3 的Sibling為何？右子點為何？
    - Node 2 的左右子點和父點編號為何？
    - Node 4 的Grand Parent為何？



- 有一個Complete B.T., 共有1000個節點 (編號：1~1000)。
  - The number of "Last parent" =_____。
  - Node 256 的Grand Parent =____。
  - Node 347 的Sibling =_____。
  - Node 450 的左子點 =_____, 右子點 =_____, 父點 =_____。
  - Node 600 的 左子點 =________ 。
  - 樹高 = $\lceil \log_2(1000+1) \rceil$ =____。
  - 有 ________________ 個葉子節點。
  - Degree為1的Node數有__個, 編號為_____。

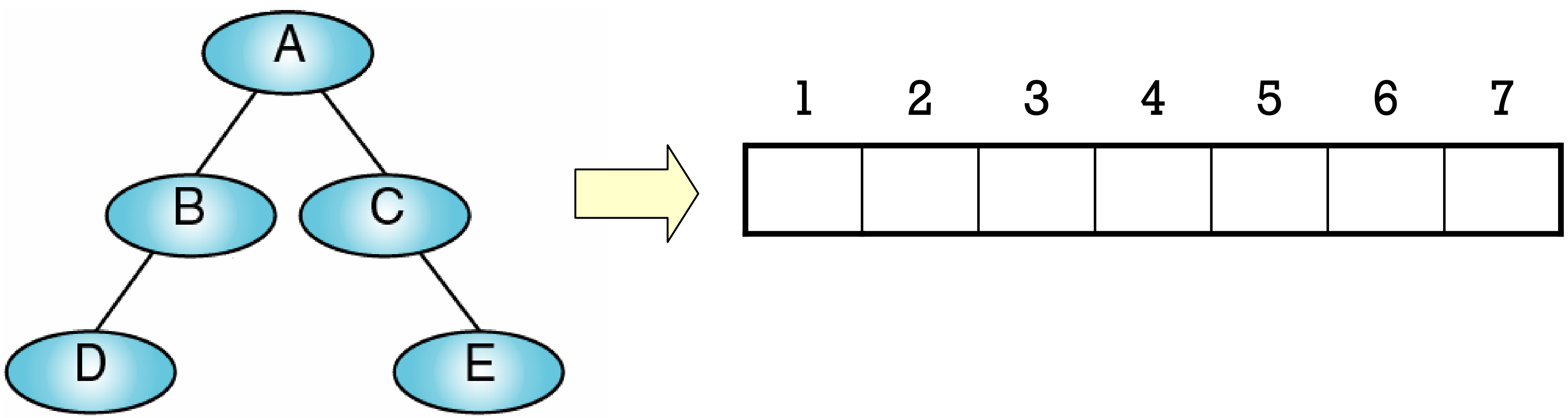
  ($\because n_0 = 500$, $n_2 = 499$。$\therefore n = n_0+n_1+n_2 \Rightarrow n_1 = 1$)

Binary Tree Structure

- 有兩種實作方法:
  - 利用Link List
  - 利用Array

利用 Array

- 作法:
  - 假設此二元樹的高度為H，則準備一個大小為2^H-1的一維陣列 A[1 ... 2^H-1] 以相對於Full B.T.之節點編號，並將之一一對應填入。
  - 例: 此二元樹的高度為3，則準備一個大小為 **2^3-1 = 7** 的一維陣列



- 優點:
  - 對於Full B.T.之儲存，完全沒有浪費空間
  - 易於取得某node之左、右子節點及父節點之資料
    - 當某個節點其編號為i，則:
      - 其左兒子編號為2i，若 2i > n，則左兒子不存在
      - 其右兒子編號為2i+1，若 2i+1 > n，則右兒子不存在
      - 其父點編號為[i/2] ([ ]: 無條件捨位取整數)；若 [i/2] < 1，則父節點不存在
- 缺點:
  - 節點增刪不易
    - ∵空間要費力移動。若空間不夠用時，需重新宣告Array
  - 對於Skewed B.T.之儲存，極度浪費空間

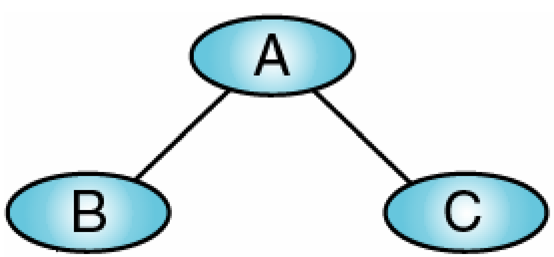
- 為何浪費空間？
  - 假設有一個高度為k的Skewed B.T., 則
    - 需準備一個具有______個空間之Array
    - 實際會用到___個空間
    - ∴會浪費 _______ 個空間
  - 如: 有一個 k = 10 的Skewed B.T., 需為它準備一個空間大小為1023的Array, 但實際卻只用上10個空間!!

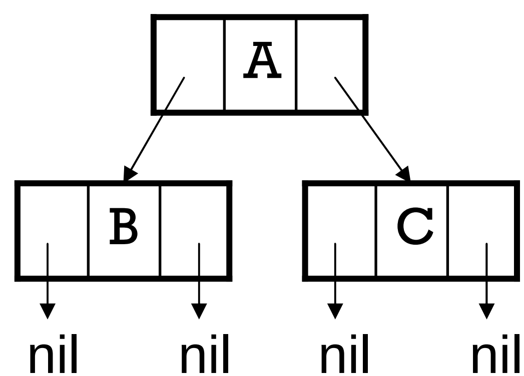
- 節點結構設計如下：

Lchild	Data	Rchild

```
Node
    leftSubTree <pointer to Node>
    data <dataType>
    rithtSubTree <pointer to Node>
End Node
```

- 例如:

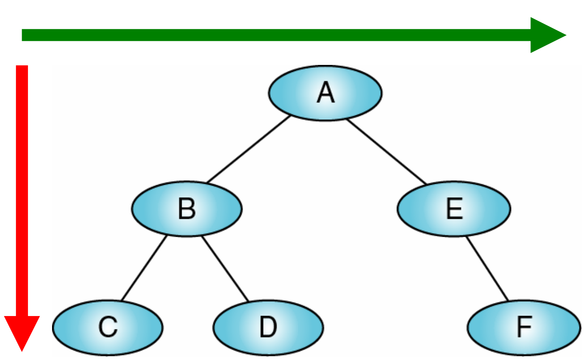




- 優點:
  - 對於Skewed B.T.之儲存較Array節省空間
  - Node之增刪容易
- 缺點:
  - 不易取得父點
    - ∵指標只用以指向兩個孩子!
  - Link空間仍浪費約一半
- 分析: 假設有n個nodes,
  - 總共的link空間: ___
  - 有用的link空間 (即: link ≠ nil): ___
  - ∴浪費的link空間 (即: link = nil): ___________

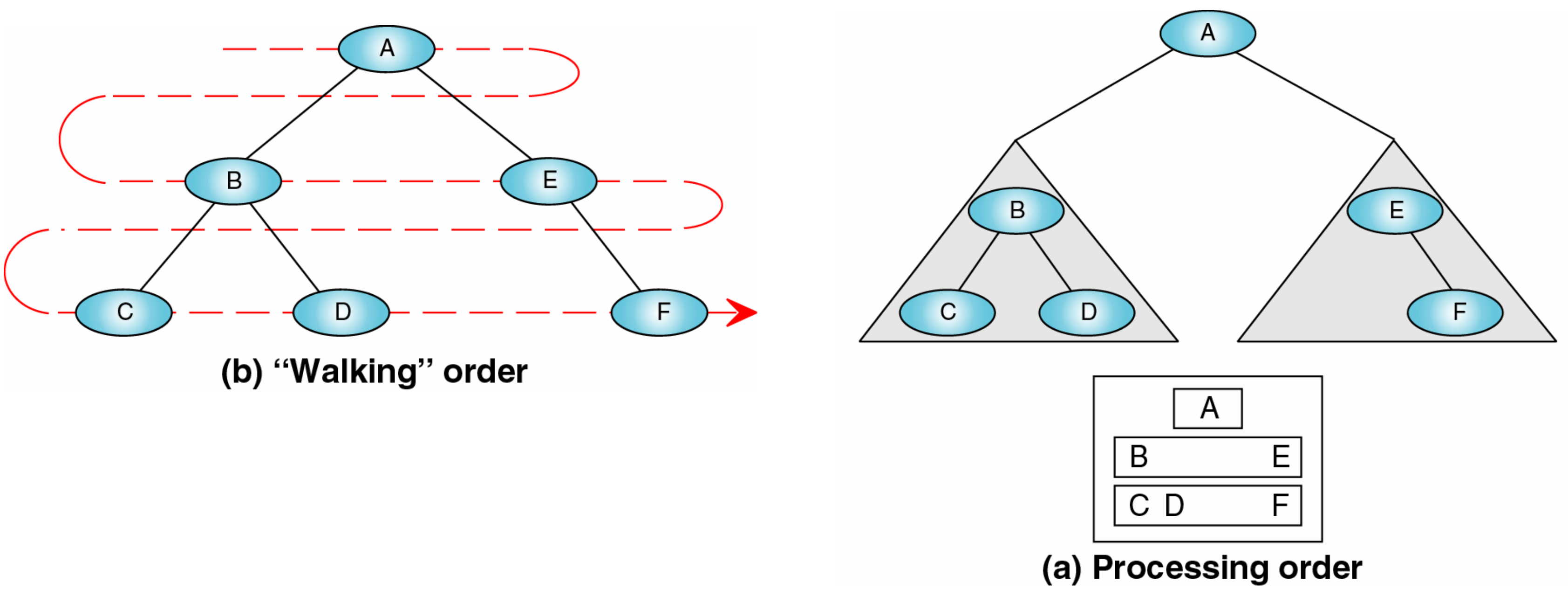
Binary Tree Traversal (二元樹追蹤)

- 二元樹追蹤主要是拜訪二元樹中每個Node資料各一次。
- 有兩個不同的追蹤方式:
  - **Depth first (深度優先; 深先)**
    - 由根節點出發，選擇某一子樹並以垂直方向由上到下處理，將其所有後裔節點拜訪完畢後，再選擇另一子樹遞迴地處理。
  - **Breadth first (廣度優先; 廣先)**
    - 由根節點出發，以水平方向由左到右處理，將所有同一level之兄弟節點拜訪完畢後，再選擇下一level之所有節點加以處理。

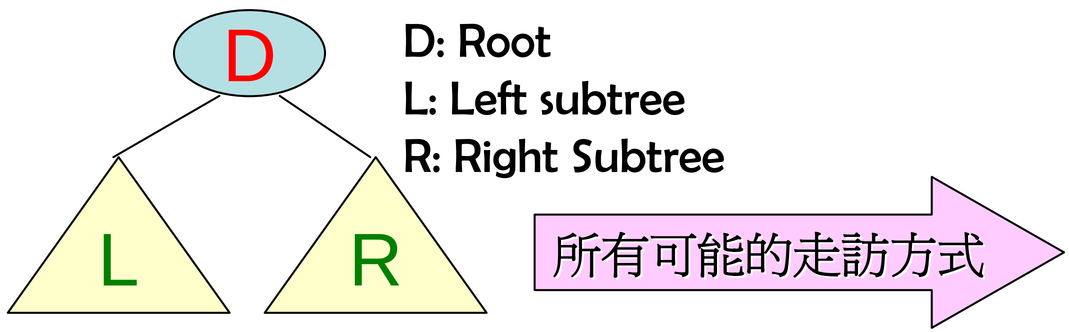


Breadth-First Traversals

- 將某一level的所有節點處理完畢後，再處理下一level的所有節點。



- 一個二元樹包括了根節點 (Root)、左子樹 (Left subtree) 與右子樹 (Right subtree)，因此可能會有 6 種不同的Depth-First Traversal 的走訪程序：



DLR
DRL
LDR
LRD
RDL
RLD

⇨若限制L一定要在R之前走訪:

- DLR: Preorder (前序)
- LDR: Inorder (中序)
- LRD: Postorder (後序)

- 看到D就**印出 (處理) 資料**!!
- 上述走訪方式皆具備**遞迴**特性。

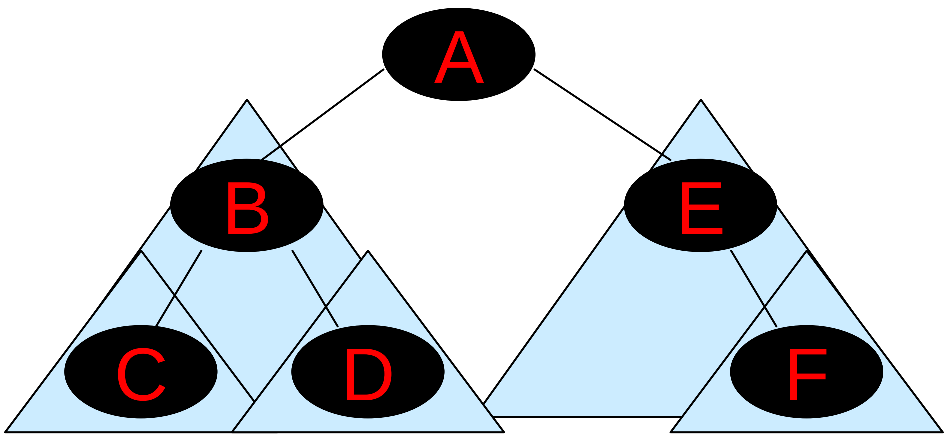
Depth-First Search的討論議題：

1. 給定一顆二元樹，則執行前、中、後序的追蹤結果為何。
2. 二元樹追蹤的遞迴演算法。
3. 給定一組中序與後序 (或：中序與前序) 的追蹤結果，如何決定出一顆唯一的二元樹。【配對問題】
4. 二元樹遞迴追蹤演算法的其它應用。

■DFS議題1、2: 二元樹的追蹤次序與遞迴演算法

- Preorder (DLR)
- Inorder (LDR)
- Postorder (LRD)

- 在前序追蹤時，**根節點**會最先被處理；再來是遞迴處理該根節點的左子樹；待左子樹之所有節點均處理完畢後，再遞迴處理該根節點之右子樹。(**DLR**)
- Ex:



Output:
A
B
C
D
E
F

Procedure *preOrder* (T: Pointer to B.T. root)

if (T is not null) //即: 樹為非空時

print (T →**data**); ⇨ D

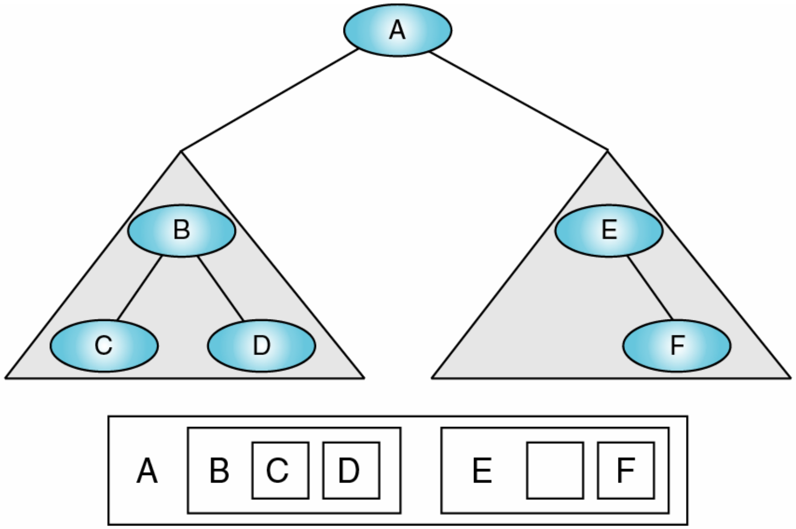
preOrder (T→leftSubtree); ⇨ L

preOrder (T →rightSubtree); ⇨ R

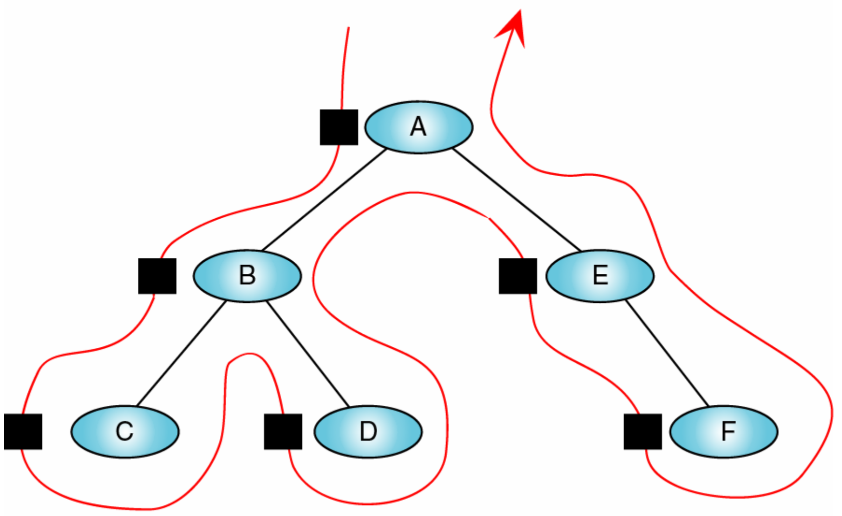
end if;

return; //由堆疊處取得應當返回之上層資訊

end preOrder.

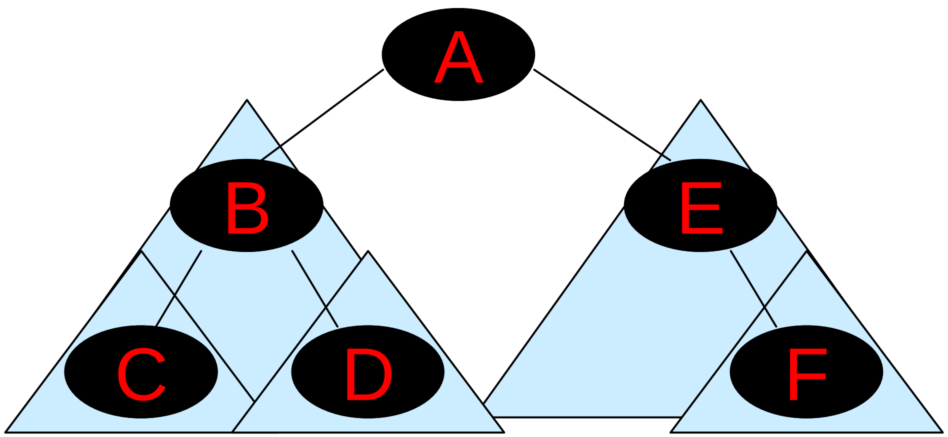


(a) Processing order



(b) "Walking" order

- 在中序追蹤時，需要最先遞迴處理根節點的左子樹；待左子樹之所有節點均處理完畢後，再處理**根節點**；最後再遞迴處理根節點之右子樹。(**LDR**)
- Ex:



Output:
C
B
D
A
E
F

Procedure *inOrder* (T: Pointer to B.T. root)

if (T is not null) //即: 樹為非空時

inOrder (T→leftSubtree); ⇨ L

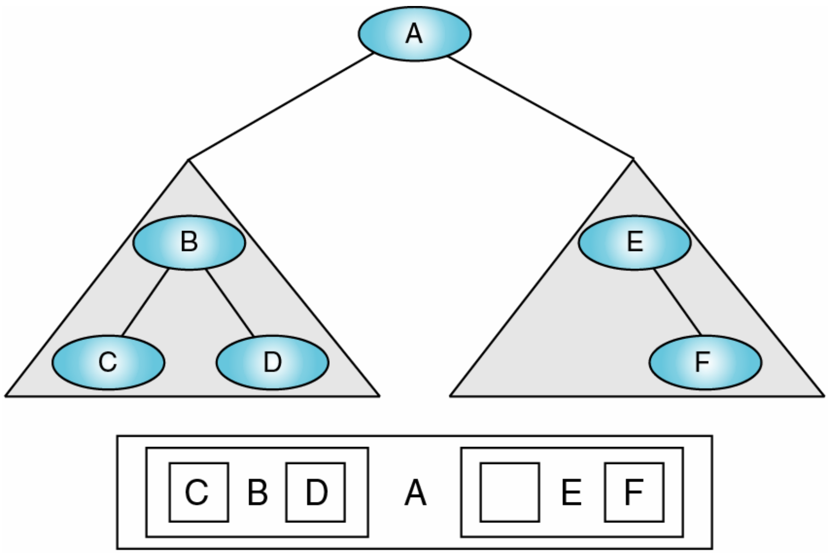
print (T → **data**); ⇨ D

inOrder (T →rightSubtree); ⇨ R

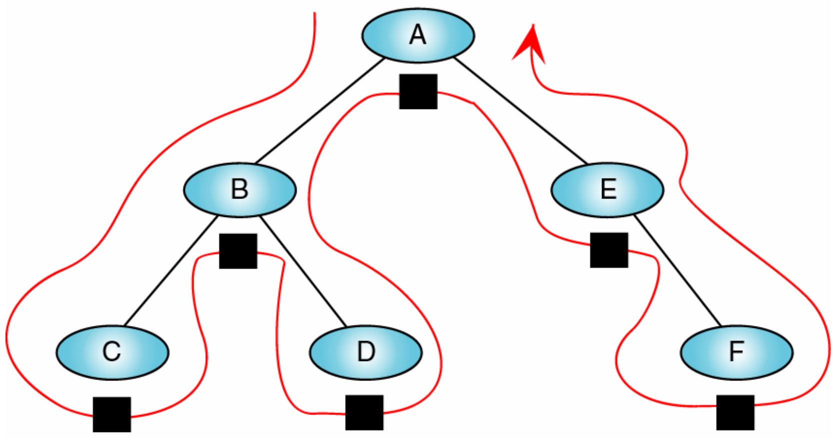
end if;

return; //由堆疊處取得應當返回之上層資訊

end preOrder.

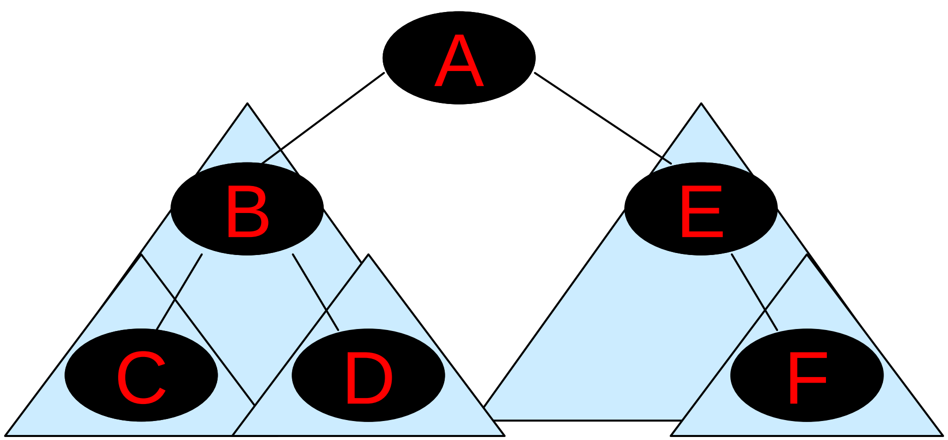


(a) Processing order



(b) "Walking" order

- 在後序追蹤時，需要最先遞迴處理根節點的<u>左子樹</u>；待左子樹之所有節點均處理完畢後，再遞迴處理根節點之<u>右子樹</u>；最後再處理**根節點**。(**LRD**)
- Ex:



<u>Output:</u>
C
D
B
F
E
A

Procedure *postOrder* (T: Pointer of B.T. root)

if (T is not null)　　//即: 樹為非空時

postOrder (T →leftSubtree);　　⇨ L

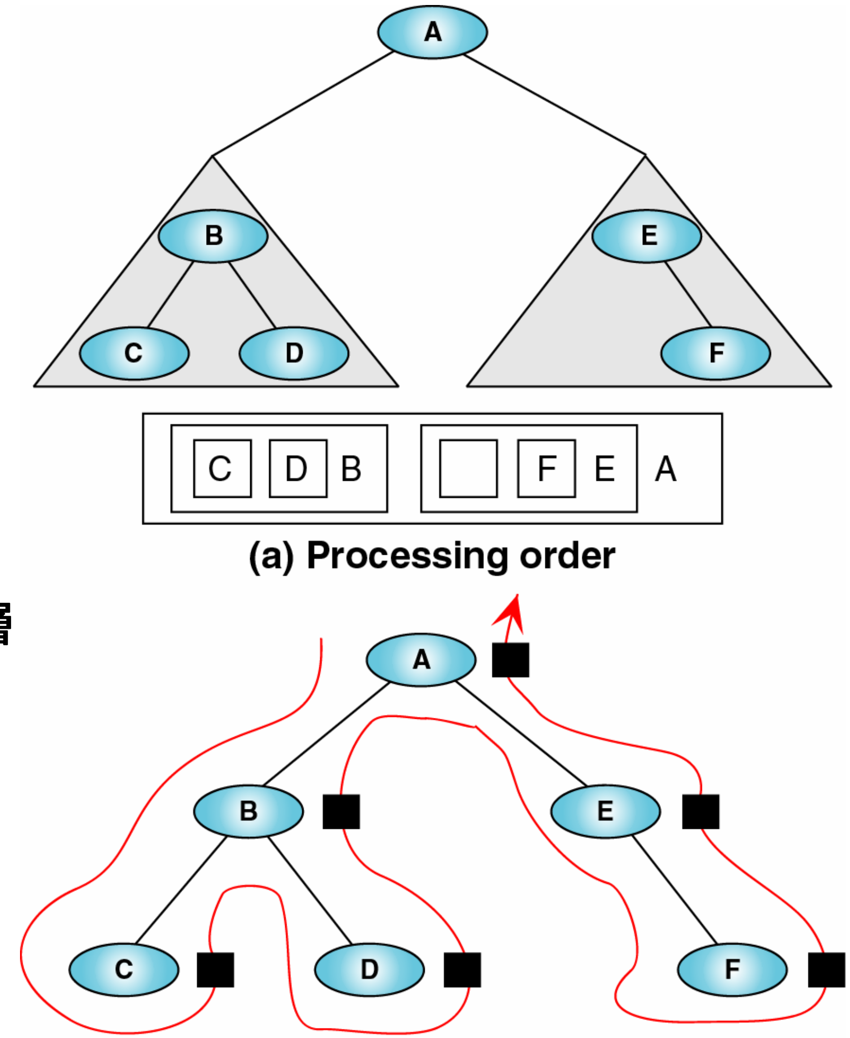
postOrder (T →rightSubtree);　　⇨ R

print (T →**data**);　　⇨ D

end if;

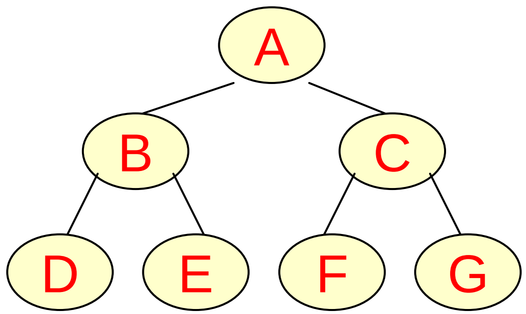
return;　　//**由堆疊處取得應當返回之上層資訊**

end preOrder.

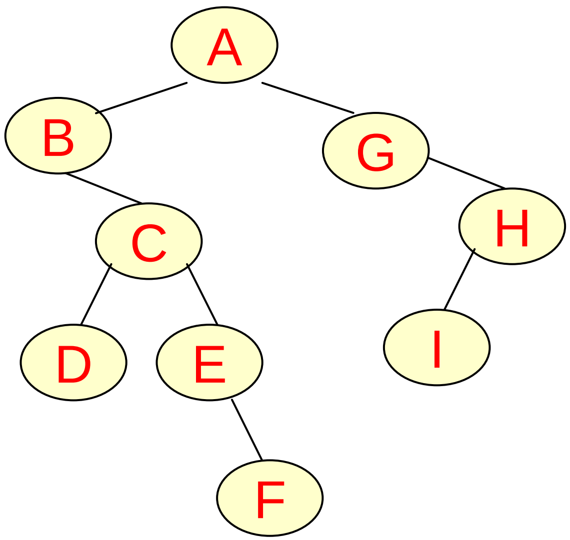


- 試求下列兩顆樹之前、中、後序之追蹤結果。

a.



b.



Ans:

a. (DLR): ABDECFG, (LDR): DBEAFCG, (LRD): DEBFGCA

b. (DLR): CABDEFGHIJ, (LDR): ADEBCGHJIF, (LRD): EDBAJIHGFC

■DFS議題3: 配對問題

- 給予 "**中序與前序**"之配對, 或是 "**中序與後序**"之配對, 必可決定**唯一的Binary Tree**, 但 "前序與後序"之配對無法決定出唯一的Binary Tree (可能會有很多組).

- 例1: 給予
  - 前序: ABCDEFGHI
  - 中序: BCAEDGHFI

則此Binary Tree為何?

(觀念: 用**前序**DLR決定root, 用中序畫tree)

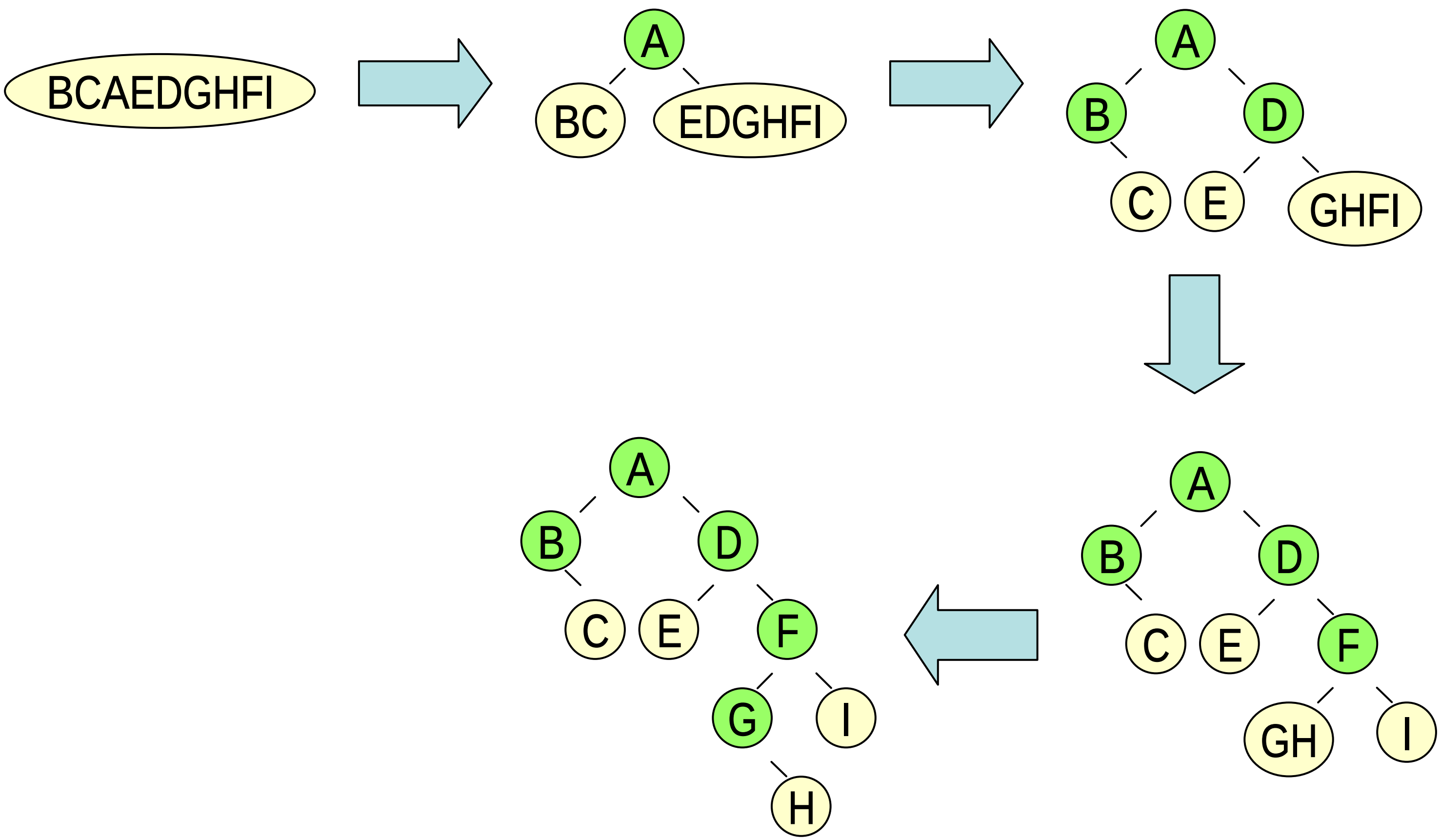
- 例2: 給予
  - 後序: EDBAJIHGFC
  - 中序: ADEBCGHJIF

則此Binary Tree為何?

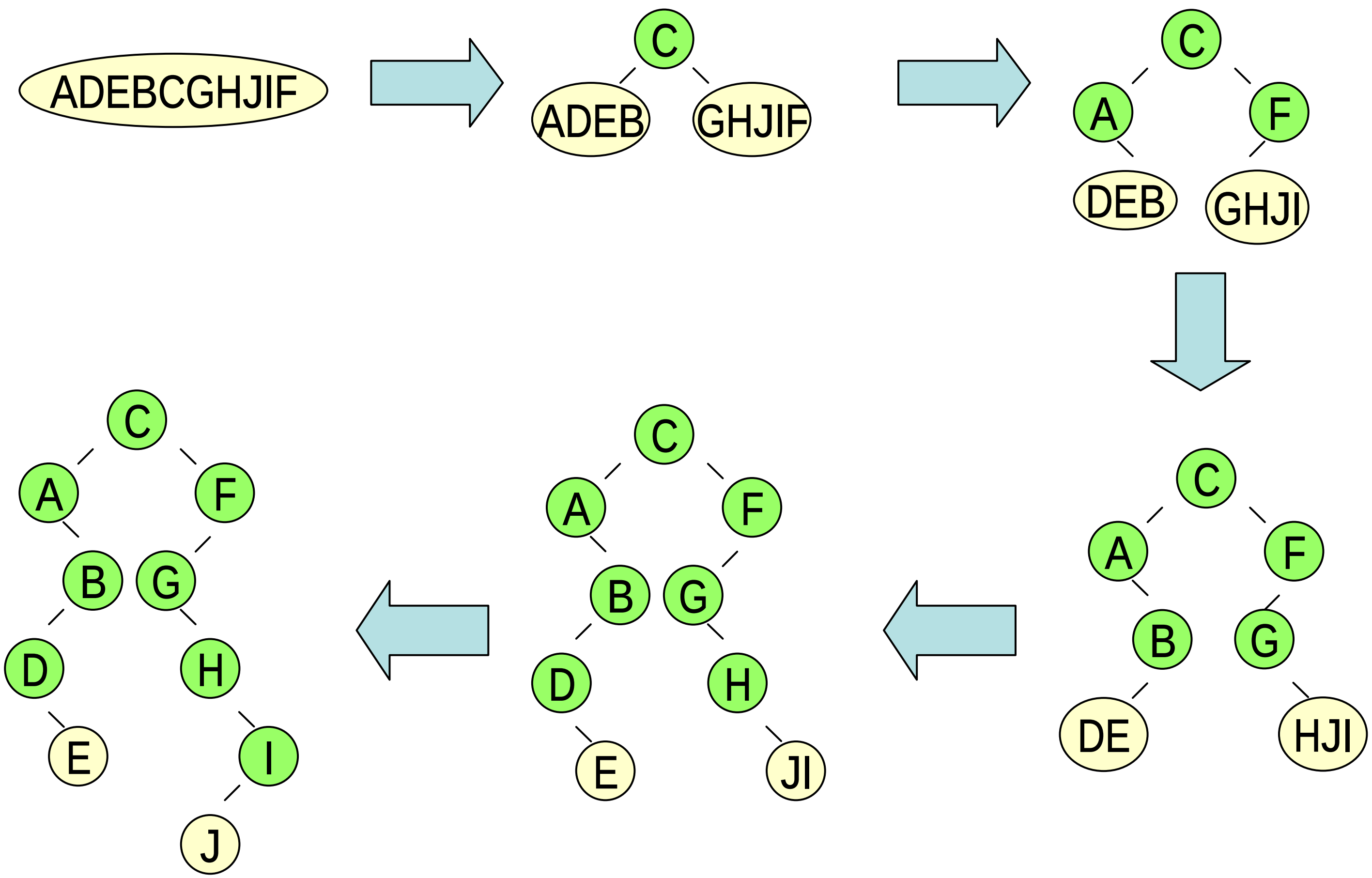
(觀念: 用**後序**LRD決定root, 用中序畫tree)

例 1

- 前序: ABCDEFGHI (DLR)
- 中序: BCAEDGHFI



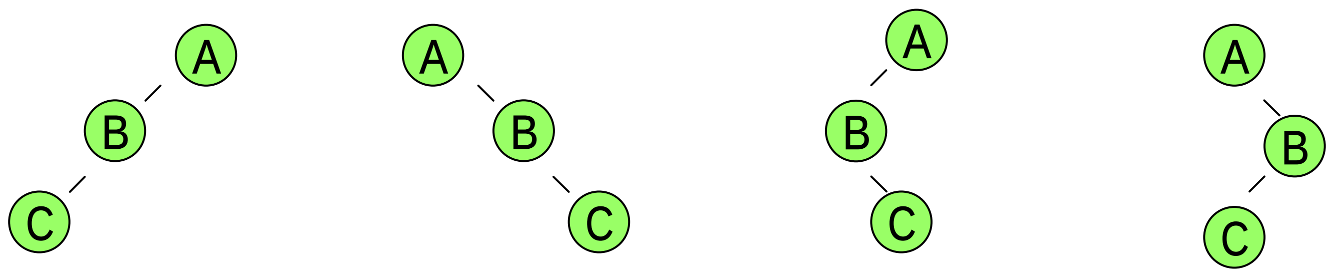
- 後序: EDBAJIHGFC (LRD)
- 中序: ADEBCGHJIF



- 例: 給予
  - 前序: ABC
  - 後序: CBA

則此Binary Tree為何?

Sol:

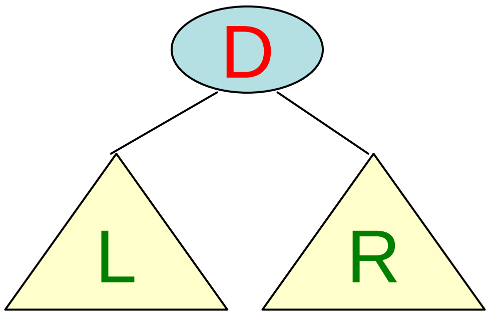


■DFS議題4: DFS遞迴演算法其它應用

- Count : 計算B.T.的節點總數
- Height : 計算B.T.的高度
- SWAP : B.T.的每一個節點之左右子樹互換
- 二元搜尋樹

- 觀念：
  - 若B.T.為空樹，則回傳結果為 0
  - 若B.T.不為空樹，則：

❸Return (n_L+n_R+1)



❶遞迴計算左子樹的節點個數n_L

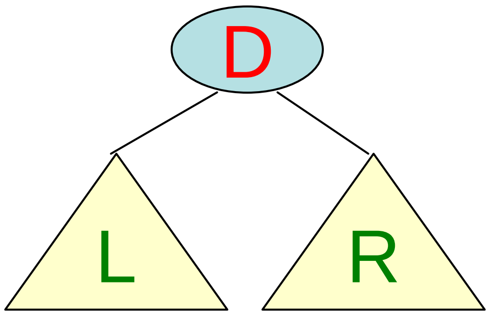
❷遞迴計算右子樹的節點個數n_R

```
Procedure Count (T: Pointer of B.T. root)
    if (T is not null) {    //即: 樹爲非空時
        nL = Count (T →leftSubtree);
        nR = Count (T →rightSubtree);
        return (nL+nR+1);
    } else
        return (0);
    end if;
    return;
end Count.
```

Height：計算B.T.的高(深)度

- 觀念：
  - 若B.T.為空樹，則回傳結果為 0
  - 若B.T.不為空樹，則：

❸Return (MAX(H_L, H_R)+1)



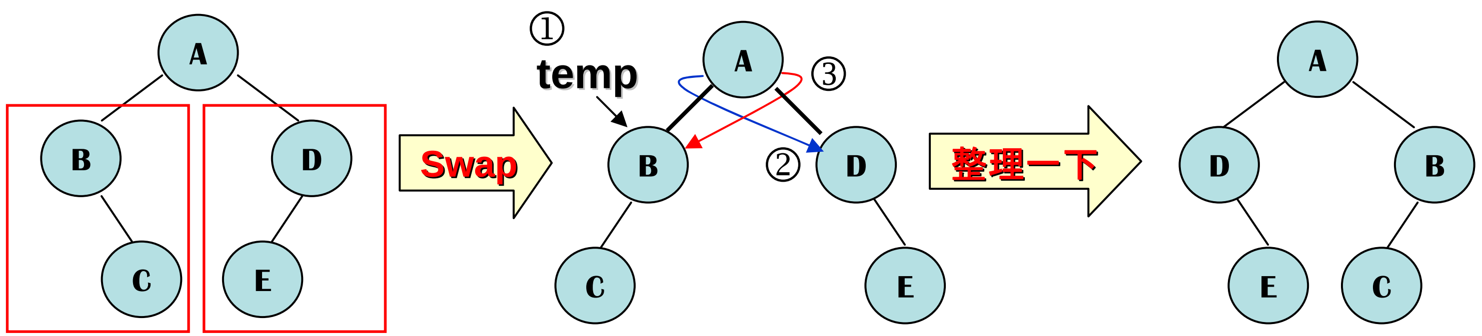
❶遞迴計算左子樹的高度 H_L

❷遞迴計算右子樹的高度 H_R

```
Procedure Height(T: Pointer of B.T. root)
    if (T is not null) {     //即: 樹爲非空時
        HL = Height(T →leftSubtree);
        HR = Height(T →rightSubtree);
        return (Max(HL, HR)+1);
     } else
         return (0);
    end if;
    return;
end Height.
```

- 觀念：
  - 若B.T.為空樹，則不需執行此工作
  - 若B.T.不為空樹，則：

❸將Root的左右子樹互換

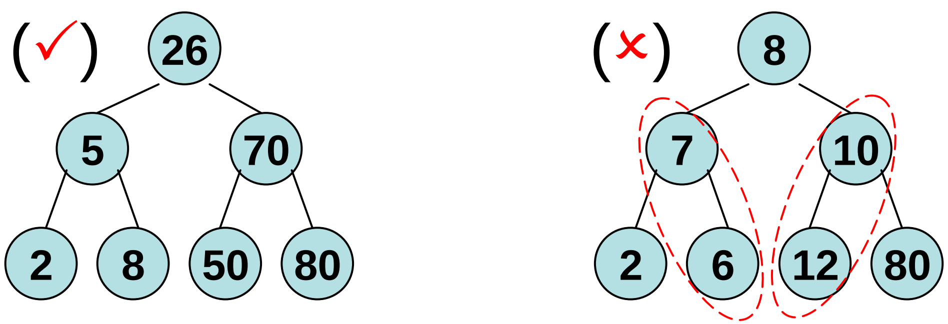


❶遞迴交換左子樹的所有節點

❷遞迴計算右子樹的所有節點

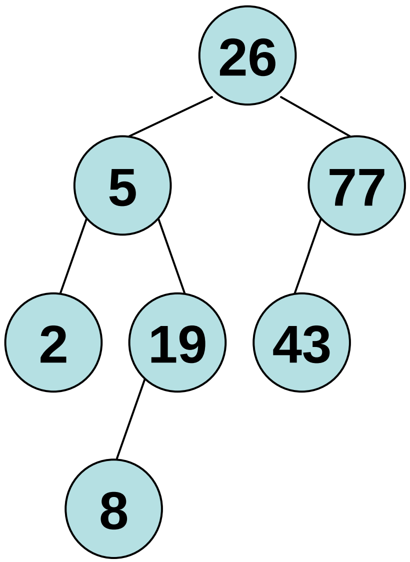
```
Procedure Swap(T: Pointer of B.T. root)
    if (T is not null) {    //即: 樹爲非空時
            Swap(T →leftSubtree);
            Swap(T →rightSubtree);
        ① temp = (T →leftSubtree);
        ② T→leftSubtree = T→rightSubtree;
        ③ T→rightSubtree = temp;
    }end if;
    return;
end Swap.
```

- 可應用於資料的排序 (Sort) 與搜尋 (Search)
- Def: 為一個Binary Tree；可以為空，若不為空則需滿足：
  - 左子樹所有Node鍵值均小於Root鍵值
  - 右子樹所有Node鍵值均大於Root鍵值
  - 左、右子樹亦是Binary Search Tree
- 範例：



如何建立B.S.T

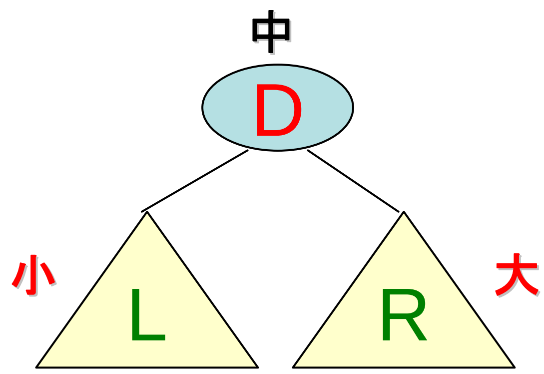
- 依據資料輸入的順序，執行 "**Insert a node into B.S.T**"之工作：
  - 輸入的資料皆與**樹根**比，比樹根小就往左子樹放，比樹根大就往右子樹放。
- 例：26, 5, 77, 43, 19, 2, 8



如何利用Binary Search Tree對資料進行排序 (Sort)

- 先將資料建成二元搜尋樹
- 依**Inorder**追蹤，即可得出由小到大的排序結果

觀念：

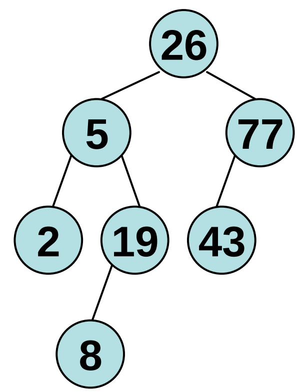


先走左，再走中，後走右

- 例：請將下列資料由小到大排序：26, 5, 77, 43, 19, 2, 8

Ans：

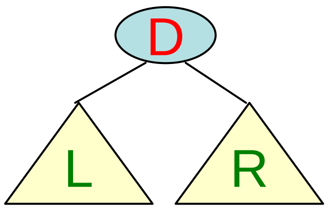
- 先將資料建成二元搜尋樹



- 依**Inorder**追蹤，即可得出**由小到大**的排序結果
  - 2, 5, 8, 19, 26, 43, 77

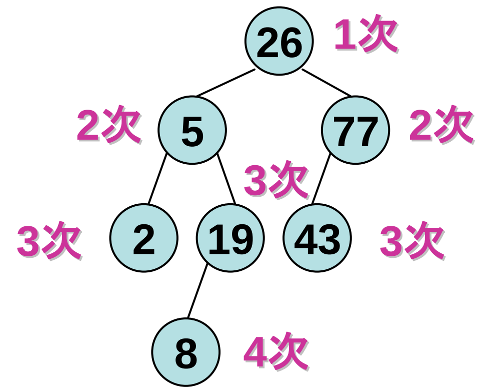
如何利用Binary Search Tree對資料進行搜尋 (Search)

- 演算法設計觀念：
  - 如果在B.S.T.有找到想要的資料，則 return Data；否則就 return Not Found
  - 假設要找的資料是k，拿k跟樹根的資料 t 做比較：
    - 若是 “=”，表示找到資料
    - 若是 “<”，表示要往左子樹找 (遞迴)
    - 若是 “>”，表示要往右子樹找 (遞迴)
  - 似 “前序追蹤” (DLR)



```
Procedure Search (T: Pointer to B.T. root, k: 要找的資料)
    if (T is not null) {                                //即: 樹為非空時
        if (k = T →data)
            return (T →data);                           ⇨ D
        else if (k < T →data)
                Search (T→leftSubtree);                 ⇨ L
            else
                Search (T →rightSubtree);               ⇨ R
    } else
        return “Not Found”;
    end if;
    return;  //由堆疊處取得應當返回之上層
               資訊
end Search.
```

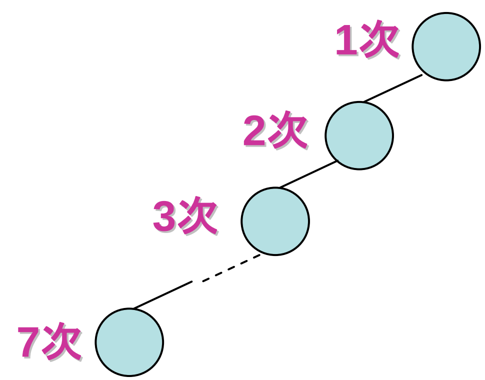
- 若有一個7個Nodes的B.S.T.如右，則其 "成功搜尋" 的平均比較次數為何?



Ans:

  - (1+2+2+3+3+3+4) / 7 = 18/7 (次)

- 若有一個7個Nodes的B.S.T.如右，則其 "成功搜尋" 的平均比較次數為何?



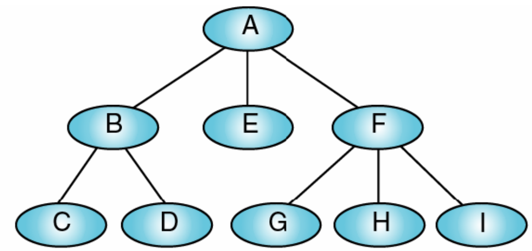
Ans:

  - (1+2+3+4+5+6+7) / 7 = 4 (次)
  - 此為B.S.T.從事資料搜尋的**最差情況**

- 結論：
  - 以B.S.T.從事資料搜尋時，其左、右子樹**愈平衡愈好**。
  - B.S.T.從事資料搜尋時，易受**資料輸入順序**的影響。

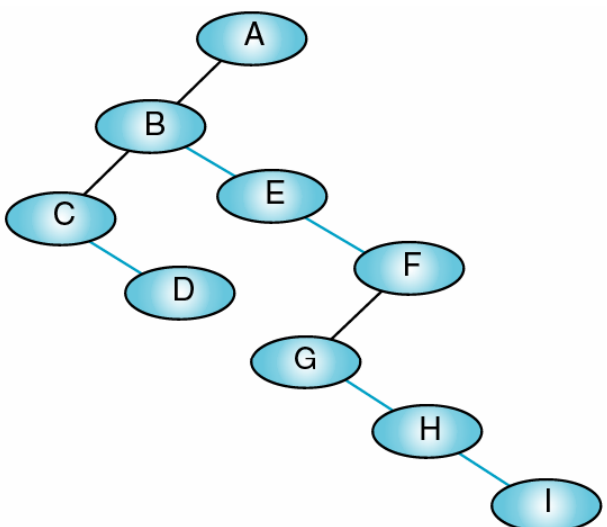
一般樹化成二元樹

- 作法:
  - 建立**Sibling之間的Link**
  - 每個節點只保留**與原最左 (leftmost) child間的link**及**Sibling間之link**，其餘父點與子點間的link皆刪除
  - 順時針**轉45度**



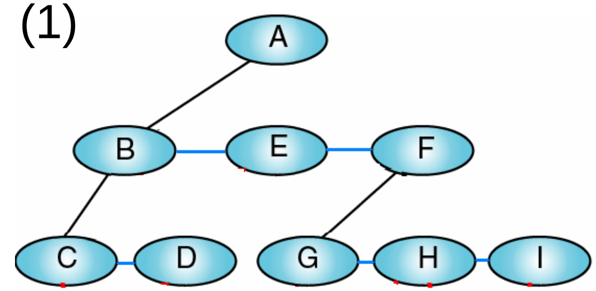
The general tree

- 【反向】二元樹化成一般樹:
  - 逆時針轉45度 (即:右兒子上拉成右兄弟)
  - 建立父點與子點之間的links
  - 刪除Sibling之間的links
- 範例:

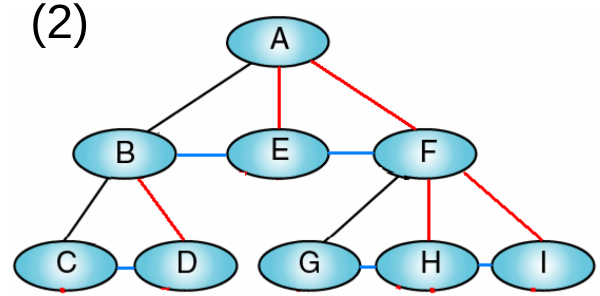


The binary tree

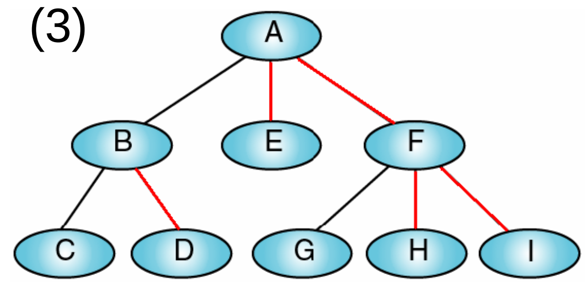
(1)



(2)



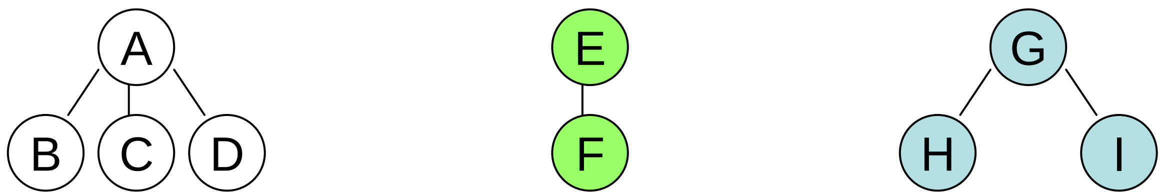
(3)



The resulting general tree

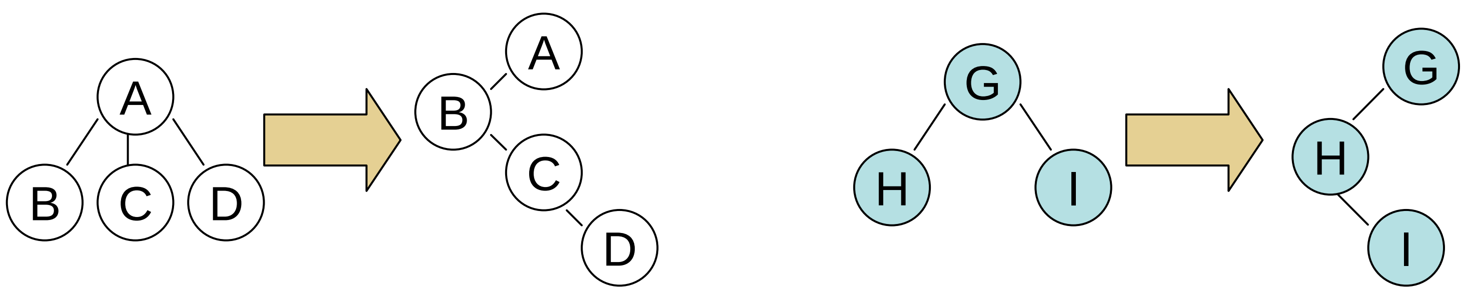
Forest 化成二元樹

- Step:
  1. 將森林中每棵Tree先化成Binary Tree
  2. 將各Binary Tree的Root以Sibling方式連結
  3. 針對這些Roots，順時針轉45度
- 例:

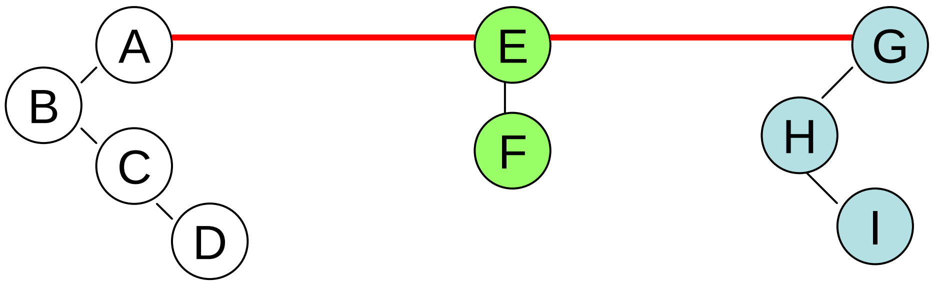


Sol:

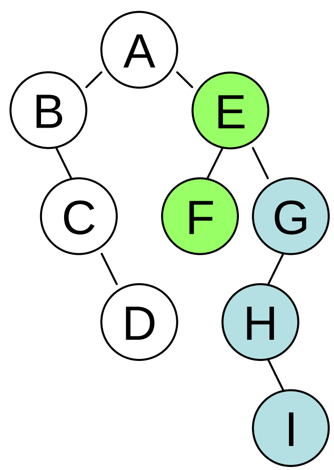
1. 將森林中每棵Tree先化成Binary Tree



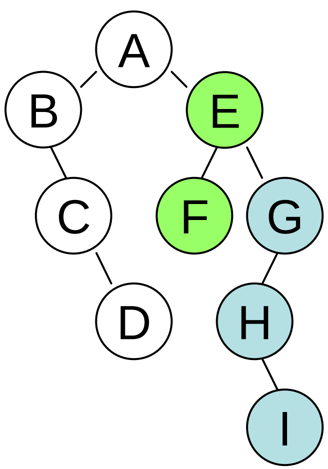
2. 將各Binary Tree的Root以Sibling方式連結

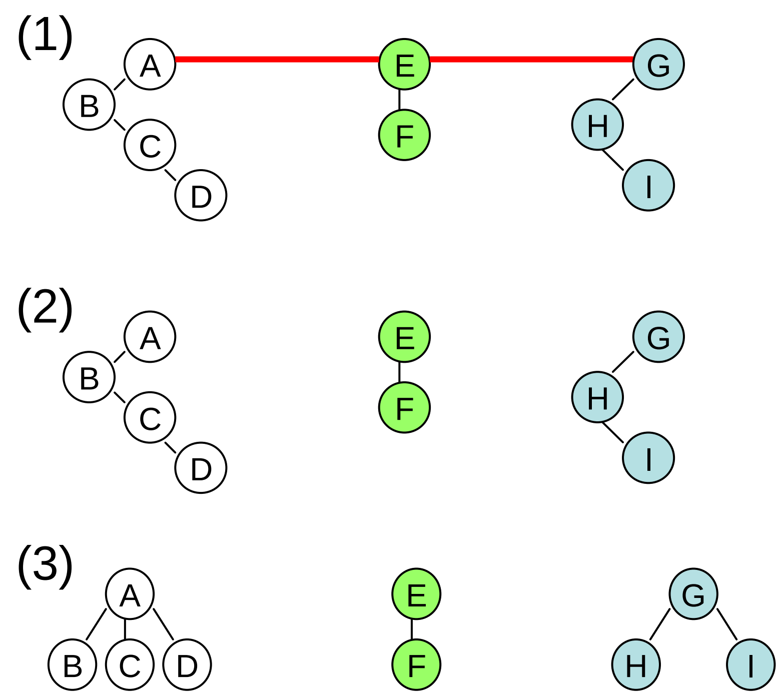


3. 針對這些Roots，順時針轉45度



- 【反向】二元樹化成森林：
  - 逆時針轉45度 (即：將Root右子樹中的所有最右節點上拉，以成為Root的右兄弟)
  - 刪除Root間之Sibling的links，以形成多個獨立的B.T.
  - 將各個B.T.化成一般樹
- 範例：

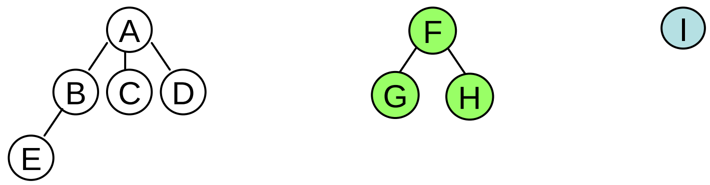




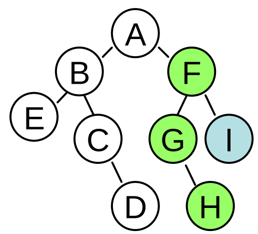
Forest 的追蹤

- Forest之Preorder、Inorder等於 "化成B.T.後，再利用B.T.的Preorder、Inorder追蹤"。
  - **Forest的Preorder**：先拜訪第一顆樹的Root，再由左至右依序分別拜訪第一顆樹的子樹；接著再拜訪第二顆樹的Root，再由左至右依序分別拜訪第二顆樹的子樹；…
  - **Forest的Inorder**：先由左至右依序分別拜訪第一顆樹的子樹，再拜訪第一顆樹的Root；接著再由左至右依序分別拜訪第二顆樹的子樹，再拜訪第二顆樹的Root；…
- Forest之Postorder不等於 "化成B.T.後，再利用B.T.的Postorder追蹤"。
  - **Forest的Postorder**：先由左至右依序分別拜訪第一顆樹的子樹，接著再由左至右依序分別拜訪第二顆樹的子樹…。當拜訪完所有的子樹後，再由最後一顆樹的Root往回追蹤。

- 範例:



  - Inorder, Preorder:



Pre: ABECDFGHI

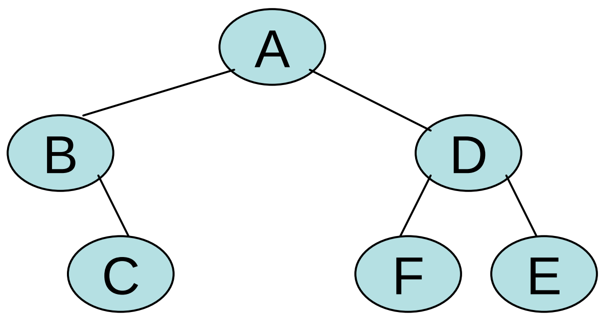
In: EBCDAGHFI

  - Postorder:EDCBHGIFA

■ Thread Binary Tree (引線二元樹)

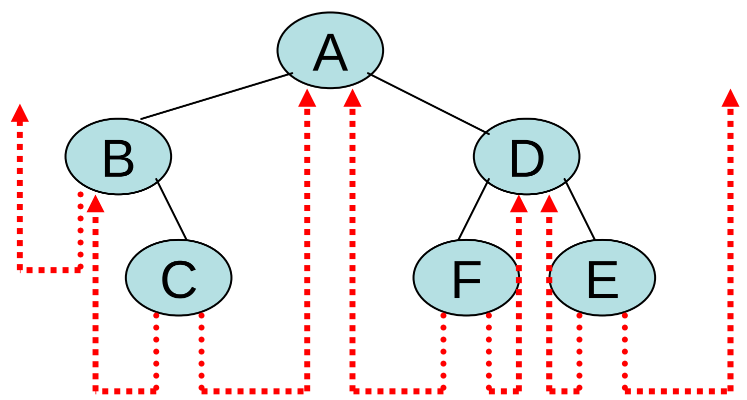
- Def: 一個二元樹具有n個節點，以link list表示，會有n+1條空links。為了充分利用這些空links，將這些links**改指向其它nodes**，此種binary tree稱為Thread Binary Tree。
- 一般而言，以中序引線二元樹居多。
- 規定:
  - 若x→Lchild為nil，則將x→Lchild改指向x在中序順序的**前**一個node。
  - 若x→Rchild為nil，則將x→Rchild改指向x在中序順序的**後**一個node。
- 優點:
  - 充份利用空的Link空間
  - 不需利用遞迴 (即:不需要Stack的支援)，即可得到中序式，同時可立即知道某節點的中序後繼者與先行者。
  - 可簡化Inorder traversal

例: 給予一個二元樹如下，請繪出其Thread B.T.



Ans:

先寫出中序式: B C A F D E



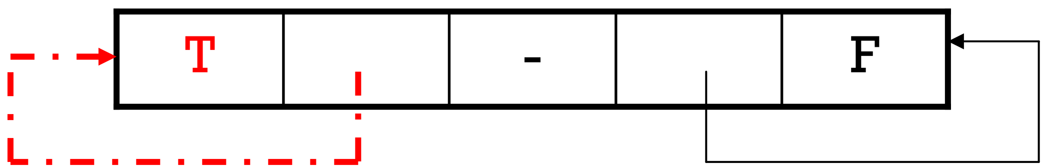
①

Boolean				Boolean
Left Thread	Lchild	Data	Rchild	Right Thread

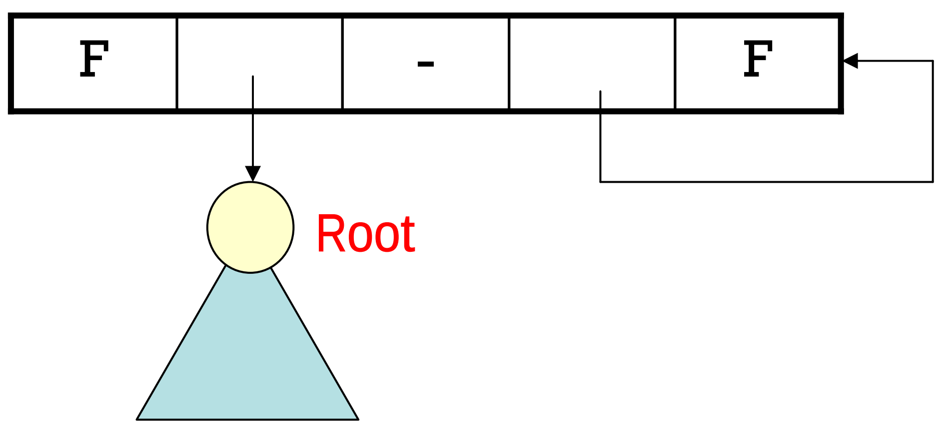
- Left Thread:
  - True: 表Lchild為左引線指標
  - False: 表Lchild為左兒子指標
- Right Thread:
  - True: 表Rchild為右引線指標
  - False: 表Rchild為右兒子指標

② 使用時，會再加入一個**Head Node (串列首)**

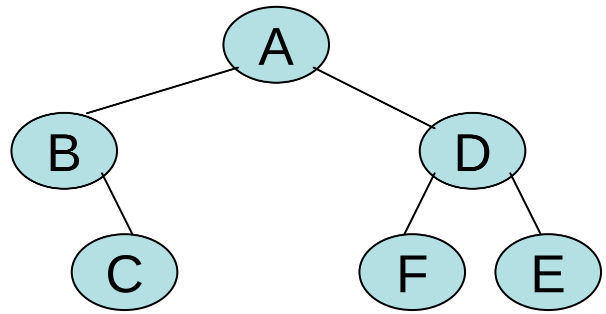
- 空樹時:



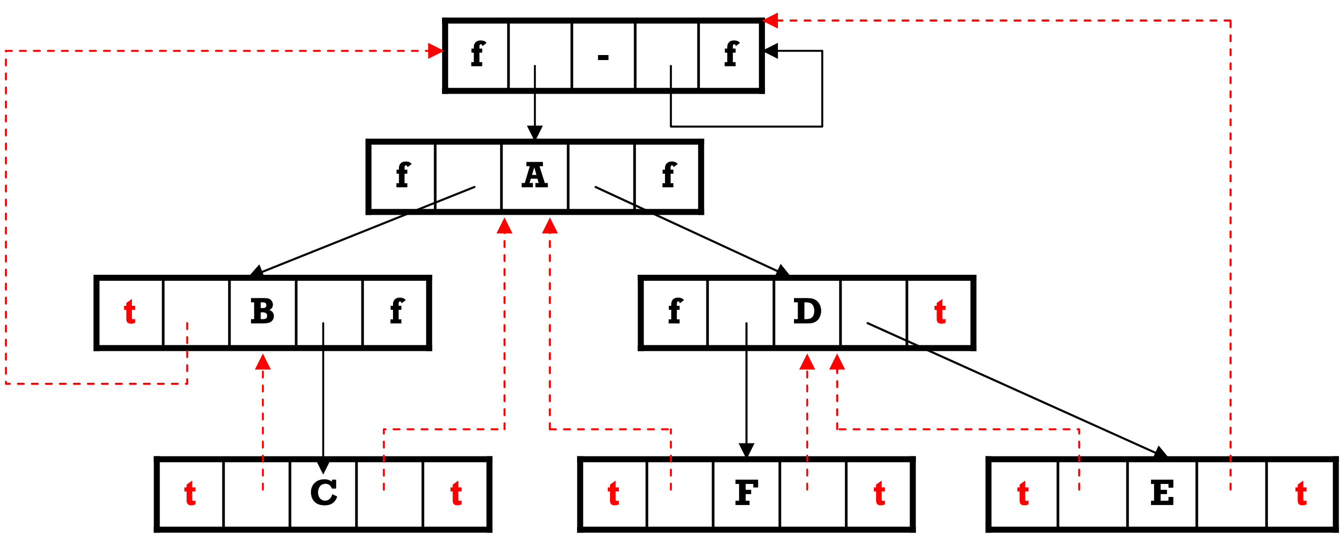
- 非空樹時:



- 將下列二元樹繪出其Thread B.T.之Data Structure.



Sol:



補　充

■二元樹三個基本定理之練習範例

- 有一棵tree，其tree degree為5，其中degree為w的node個數有w個，$1 \le w \le 5$，則leaf個數 = ?

Sol:

$n = n_0+n_1+n_2+n_3+n_4+n_5$

$= n_0+1+2+3+4+5$

$= n_0+15$

$= B+1$

$= (n_1\times 1 + n_2\times 2 + n_3\times 3 + n_4\times 4 + n_5\times 5) + 1$

$= (1\times 1 + 2\times 2 + 3\times 3 + 4\times 4 + 5\times 5) + 1$

$\therefore n_0+15 = (1+4+9+16+25)+1 = 56$

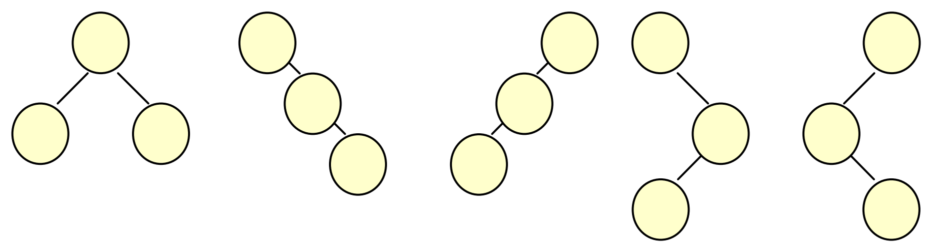
⇨ $n_0 = 41$.

B: Branch總數 (即: 有用的**link**數，**n-1**)
⇨ **n = B+1**

- [定理]: n個節點可以形成 $\underline{\frac{1}{n+1}\binom{2n}{n}}$ 棵不同的二元樹。
- 範例:
  - 3個節點可以形成幾個不同的 B.T., 請一一繪出。

Sol:

$n=3 \Rightarrow \frac{1}{3+1}\binom{6}{3}=5$



  - A, B, C三個資料, 其中A<B<C, 可以形成幾個不同的B.T.？試繪出。

Sol: 5顆 (同上題。Hint: 左節點比root小，右節點比root大)